arXiv:submit/0806879 [cs.LG] 22 Sep 2013

Stochastic Bound Majorization

Anna Choromanska
Department of Electrical Engineering
Columbia University
New York
aec2163@columbia.edu

Tony Jebara
Department of Computer Science
Columbia University
New York
jebara@cs.columbia.edu

Abstract

Recently a majorization method for optimizing partition functions of log-linear models was proposed alongside a novel quadratic variational upper-bound. In the batch setting, it outperformed state-of-the-art first- and second-order optimization methods on various learning tasks. We propose a stochastic version of this bound majorization method as well as a low-rank modification for high-dimensional data-sets. The resulting stochastic second-order method outperforms stochastic gradient descent (across variations and various tunings) both in terms of the number of iterations and computation time till convergence while finding a better quality parameter setting. The proposed method bridges first- and second-order stochastic optimization methods by maintaining a computational complexity that is linear in the data dimension and while exploiting second order information about the pseudo-global curvature of the objective function (as opposed to the local curvature in the Hessian).

1 Introduction

Stochastic learning algorithms are of central interest in machine learning due to their simplicity and, as opposed to batch methods, their low memory and computational complexity requirements [1, 2, 3]. For instance, stochastic learning algorithms are commonly used to train deep belief networks (DBNs) [4, 5] which perform extremely well on tasks involving massive data-sets [6, 7]. Stochastic algorithms are also broadly used to train Conditional Random Fields (CRFs) [8], solve maximum likelihood problems [9] and perform variational inference [10].

Most stochastic optimization approaches fall into two groups: first-order methods and second-order methods. Popular first-order methods include stochastic gradient descent (SGD) [11] and its many extensions [12, 13, 14, 15, 16, 17]. These methods typically have low computational cost per iteration (such as $\mathcal{O}(d)$ where d is the data dimensionality) and either sub-linear (most stochastic gradient methods) or linear (as shown recently in [12]) convergence rate which makes them particularly relevant for large-scale learning. Despite its simplicity, SGD has many drawbacks: it has slow asymptotic convergence to the optimum [8], it has limited ability to handle certain regularized learning problems such as l_1-regularization [18], it requires step-size tuning and it is difficult to parallelize [19]. Many works have tried to incorporate second-order information (i.e. Hessian) into the optimization problem to improve the performance of traditional SGD methods. A straightforward way of doing so is to simply replace the gain in SGD with the inverse of the Hessian matrix which, when naively implemented, induces a computational complexity of $\mathcal{O}(d^3)$. This makes the approach impractical for large problems. The trade-offs in large-scale learning for various prototypical batch and stochastic learning algorithms are conveniently summarized in [20]). Therein, several new methods are developed, including variants of Newton's method which use both gradient and Hessian information to compute the descent direction. By carefully exploring different first- and second-order techniques, the overall computational complexity of optimization can be reduced as in the Stochastic Meta-Descent (SMD) algorithm [21]. Although it still uses the gradient direction to converge, SMD also efficiently exploits certain Hessian-vector products to adapt the gradient step-size. The algorithm is shown to converge to the same quality solution as limited-memory BFGS (LBFGS) an order of magnitude faster for CRF training [8]. There also exists stochastic versions of quasi-Newton methods like online BFGS and online LBFGS [22], the latter applicable to large-scale problems, which while using convenient size mini-batches performs comparably to a well-tuned natural gradient descent [23] on the task of training CRFs, but at the same time is more scalable. In each iteration the inverse of the Hessian, that is assumed to have no negative eigenvalues, is estimated. Computational complexity of this online LBFGS method is $\mathcal{O}(md)$ per iteration, where m is the size of the buffer used to estimate the inverse of the curvature. The method degrades (large m) for sparse data-sets. Another second-order stochastic optimization approach proposed in the literature explores diagonal approximations of the Hessian matrix or Gauss-Newton matrix[5, 24]. In some cases this approach appears to be overly simplistic [25], but turned out successful in very particular applications, i.e. for learning with linear Support Vector Machines [24]. There is also a large body of work on stochastic second-order methods particularly successful in training deep belief network like Hessian-free optimization [25]. Finally, there are also many hybrid methods using existing stochastic optimization tools as building blocks and merging them to obtain faster and more robust learning algorithms [9, 26].

This paper contributes to the family of existing second-order stochastic optimization methods with a new algorithm that is using a globally guaranteed quadratic bound with a curvature different than the Hessian. Therefore our approach is not merely a variant of Newton's method. This is a stochastic version of a recently proposed majorization method [27] which performed maximum (latent) conditional likelihood problems more efficiently than other state-of-the-art first- and second- order batch optimization methods like BFGS, LBFGS, steepest descent (SD), conjugate gradient (CG) and Newton. The corresponding stochastic bound majorization method is compared with a well-tuned SGD with either constant or adaptive gain in l_2-regularized logistic regression and turns out to outperform competitor methods in terms of the number of iterations, the convergence time and even the quality of the obtained solution measured by the test error and test likelihood.

2 Preliminaries

A staggering number of machine learning and statistics frameworks involve linear combinations or cascaded linear combinations of soft-maximum functions:

$$s(\boldsymbol{\theta}) = \sum_{j=1}^{t} \gamma_i \log \sum_{y} h(y) \exp(\boldsymbol{\theta}^\top \mathbf{f}(y)),$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ is a parameter vector, $\mathbf{f} : \Omega \to \mathbb{R}^d$ is any vector-valued function mapping an input y to some arbitrary vector (we assume Ω is finite and $|\Omega| = n$ is enumerable), t is the size of the data-set and γ_i is some non-negative weight. These functions emerge in multi-class logistic regression, CRFs [28], hidden variable problems, DBNs [29], discriminatively trained speech recognizers [30] and maximum entropy problems [31]. For simplicity, we focus herein on the CRF training problem in particular. CRFs use the density model:

$$p(y|x_j, \boldsymbol{\theta}) = \frac{1}{Z_{x_j}(\boldsymbol{\theta})} h_{x_j}(y) \exp(\boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y)),$$

where $\{(x_1, y_1), \ldots, (x_t, y_t)\}$ are *iid* input-output pairs and $Z_{x_j}(\boldsymbol{\theta})$ is a partition function: $Z_{x_j}(\boldsymbol{\theta}) = \sum_{y \in \Omega_j} h_{x_j}(y) \exp(\boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y))$. Following the maximum likelihood approach, the objective function to maximize in this setting is:

$$J(\boldsymbol{\theta}) = \sum_{j=1}^{t} \left[\log \frac{h_{x_j}(y_j)}{Z_{x_j}(\boldsymbol{\theta})} + \boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y_j) \right] - \frac{\lambda}{2} \|\boldsymbol{\theta}\|^2, \tag{1}$$

where λ is a regularization hyper-parameter. Let $J(\boldsymbol{\theta}) = \sum_{j=1}^{t} J_j(\boldsymbol{\theta})$, where $J_j(\boldsymbol{\theta}) = \log \frac{h_{x_j}(y_j)}{Z_{x_j}(\boldsymbol{\theta})} + \boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y_j) - \frac{\lambda}{2t} \|\boldsymbol{\theta}\|^2$. For large numbers of data points t, and potentially large dimensionality d, summations in Equation 1 need not be handled in a batch form, but rather, can be processed stochastically or semi-stochastically. We next review the most commonly used stochastic algorithm, SGD, which will be a key comparator for our stochastic bound majorization algorithm.

2.1 Stochastic gradient descent methods

Batch gradient descent updates the parameter vector $\boldsymbol{\theta}$ after seeing the entire training data-set using the following formula:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta\boldsymbol{\mu}(\boldsymbol{\theta}) = \boldsymbol{\theta} - \eta\sum_{j=1}^{t}\boldsymbol{\mu}_j(\boldsymbol{\theta}),$$

where $\boldsymbol{\mu}(\boldsymbol{\theta}) = \bigtriangledown_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$, $\boldsymbol{\mu}_j(\boldsymbol{\theta}) = \bigtriangledown J_j(\boldsymbol{\theta})$ and η is typically chosen via line search. In contrast, stochastic gradient descent updates the parameter vector $\boldsymbol{\theta}$ after seeing each training data point (resp. each mini-batch of data points) as follows:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta_i\boldsymbol{\mu}_j(\boldsymbol{\theta}_i), \qquad \text{resp. } \boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta_i\sum_{j=1}^{m}\boldsymbol{\mu}_j(\boldsymbol{\theta}_i),$$

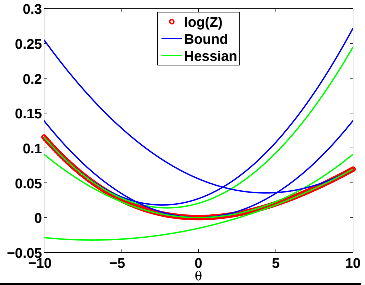
where $\boldsymbol{\theta}_i$ is the current parameter vector, η_i is the current gain or step-size, and $m << t$ is the size of the mini-batch. We assume a data point is randomly selected and take its index to be j. We intentionally index $\boldsymbol{\theta}$ with i and $i+1$ to emphasize that the update is done after seeing single example (resp. mini-batch of examples). For the batch method, we do not index $\boldsymbol{\theta}$ since the update is done after passing through the entire data-set (epoch). In the experimental section we explore two existing variants of stochastic gradient descent: one with constant gain (SGD) and one with adaptive gain (ASGD). For the latter, we consider two strategies for modifying the gain and present the results for the better one (in Section 5 the absence of a τ value indicates that the second strategy is better since it does not require a τ parameter):

- $\eta_i = \frac{\tau}{\tau+i}\eta_0$
- $\eta_i = \frac{\eta_0}{i}$, where $\eta_0, \tau > 0$ are tuning parameters.

2.2 Batch bound majorization algorithm

Algorithm 1 Batch Bound

```
Input Parameters $\tilde{\boldsymbol{\theta}}, \mathbf{f}(y), h(y)\ \forall y \in \Omega$
Init $z \to 0^+, \mathbf{g} = \mathbf{0}, \boldsymbol{\Sigma} = z\mathbf{I}$
For each $y \in \Omega$ {
        $\alpha = h(y)\exp(\tilde{\boldsymbol{\theta}}^\top \mathbf{f}(y))$;  $\mathbf{l} = \mathbf{f}(y) - \mathbf{g}$
        $\beta = \frac{\tanh(\frac{1}{2}\log(\alpha/z))}{2\log(\alpha/z)}$;  $\kappa = \frac{\alpha}{z+\alpha}$
        $\boldsymbol{\Sigma}$  $+= \beta\mathbf{l}\mathbf{l}^\top$
        $\mathbf{g}$  $+= \kappa\mathbf{l}$
        $z$  $+= \alpha$    }
Output $z, \mathbf{g}, \boldsymbol{\Sigma}$
```



Theorem 1 *Algorithm 1 finds $z, \mathbf{g}, \boldsymbol{\Sigma}$ such that $z\exp(\frac{1}{2}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top\boldsymbol{\Sigma}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}) + (\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top\mathbf{g})$ upper-bounds $Z(\boldsymbol{\theta}) = \sum_y h(y)\exp(\boldsymbol{\theta}^\top\mathbf{f}(y))$ for any $\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}, \mathbf{f}(y) \in \mathbb{R}^d$ and $h(y) \in \mathbb{R}^+$ for all $y \in \Omega$.*

The stochastic bound majorization algorithm proposed in this paper is a stochastic variant of the batch method described in [27]. The bound is depicted in Theorem 1. The figure near Algorithm 1 shows typical bounds the algorithm recovers (in blue) and also shows (in green) examples what happens when the bound's $\boldsymbol{\Sigma}$ matrix is replaced by the Hessian matrix which yields a second-order approximation to the function. These tight quadratic bounds facilitate majorization: solving an optimization problem by iteratively finding the optima of simple bounds on it [32] as popularized by the Expectation-Maximization (EM) algorithm [33]. Majorization was shown to achieve faster and monotonically convergent performance in CRF learning and maximum latent conditional likelihood problems with a clear advantage over state-of-the-art first- and second-order batch methods [27]. The batch bound majorization algorithm applied to Equation 1 updates the parameter vector $\boldsymbol{\theta}$ after seeing the entire training data-set using the following formula:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta\boldsymbol{\Sigma}^{-1}(\boldsymbol{\theta})\mu(\boldsymbol{\theta}), \tag{2}$$

where $\mathbf{\Sigma}(\boldsymbol{\theta}) = \sum_{j=1}^{t} \mathbf{\Sigma}_j(\boldsymbol{\theta}) + \lambda\mathbf{I}$, $\boldsymbol{\mu}(\boldsymbol{\theta}) = \sum_{j=1}^{t} \boldsymbol{\mu}_j(\boldsymbol{\theta}) = \sum_{j=1}^{t}(\mathbf{g}_j(\boldsymbol{\theta}) - \mathbf{f}_j(y_j)) + \lambda\boldsymbol{\theta}$. Here, each $\mathbf{\Sigma}_j$ and $\mathbf{g}_j$ is computed using Algorithm 1 (for details see [27]) and $\eta \in (0, 2)$ guarantees monotonic improvement [34] (typically we set $\eta = 1$).

3 Stochastic bound majorization algorithm

The intuition behind stochastic gradient descent is to compute the gradient over a single representative data-point rather than an iterative computation (namely a summation) over all data-points. This allows us to interleave parameter updates into the gradient computation rather than wait for it to terminate after a full epoch. We herewith extend this intuition into a bound majorization setting. Unfortunately, the update rule for majorization involves a matrix inverse rather than a simple sum across the data. We will re-cast this update rule as an iterative computation over the data-set which will then admit a stochastic incarnation.

3.1 Full-rank version

Notice that the batch update in Equation 2 is the solution of the linear system:

$$\left(\sum_{j=1}^{t} \mathbf{\Sigma}_j + \lambda\mathbf{I}\right)\boldsymbol{\delta} = \sum_{j=1}^{t} \boldsymbol{\mu}_j$$

For the ease of notation we denote $\mathbf{\Sigma}_j(\boldsymbol{\theta})$ as $\mathbf{\Sigma}_j$ and $\boldsymbol{\mu}_j(\boldsymbol{\theta})$ as $\boldsymbol{\mu}_j$. Rewrite the linear system as $\mathbf{\Sigma}\boldsymbol{\delta} = \mathbf{u}$. We then define the inverse matrix $\mathbf{M} = \mathbf{\Sigma}^{-1}$ and write the solution as $\boldsymbol{\delta} = \mathbf{M}\mathbf{u}$. Clearly, the matrix inversion cannot be performed each time we process a single data-point in a stochastic setting since a computational complexity of $\mathcal{O}(d^3)$ is prohibitive in a stochastic setting. Therefore, consider an online version of Algorithm 1 which computes the matrix inversion of $\mathbf{\Sigma}$ incrementally using the Sherman-Morrison formula $(\mathbf{\Sigma} + \mathbf{q}_i\mathbf{q}_i^\top)^{-1} = \mathbf{\Sigma}^{-1} - (\mathbf{\Sigma}^{-1}\mathbf{q}_i\mathbf{q}_i^\top\mathbf{\Sigma}^{-1})/(1 + \mathbf{q}_i^\top\mathbf{\Sigma}^{-1}\mathbf{q}_i)$. Sherman-Morrison works with the inverse matrix $\mathbf{M} = \mathbf{\Sigma}^{-1}$ instead. Initialize $\mathbf{M}_0 = \frac{1}{\lambda}\mathbf{I}$ and incrementally increase $\mathbf{M}$ using the update rule:

$$\mathbf{M}_{i+1} = \mathbf{M}_i - \frac{\mathbf{M}_i\mathbf{q}_i\mathbf{q}_i^\top\mathbf{M}_i}{1 + \mathbf{q}_i^\top\mathbf{M}_i\mathbf{q}_i}, \quad \text{where} \;\; \mathbf{q}_i = \sqrt{\beta_i}(\mathbf{f}_i - \mathbf{g}_i) = \sqrt{\beta_i}\mathbf{l}_i,$$

where the index i ranges over all rank 1 updates to the matrix $\mathbf{\Sigma}$ for all elements of Ω as well as $j = 1, \ldots, t$. Finally, the solution is obtained by multiplying $\mathbf{M}$ with $\mathbf{u}$. This avoids $\mathcal{O}(d^3)$ inversion and solves the linear system in $\mathcal{O}(tnd^2)$. Let $T = tn$. We will now reformulate the batch update from Equation 2 by using the Sherman-Morrison technique. For the ease of notation, introduce $\boldsymbol{\xi}$'s such that $\forall_{i=1,2,\ldots,T}\boldsymbol{\mu}_i = \boldsymbol{\mu}_{i-1} + \boldsymbol{\xi}_{i-1}$ (thus $\boldsymbol{\xi}_{i-1} = \kappa_{i-1}\boldsymbol{l}_{i-1} - \boldsymbol{f}_{i-1} + \lambda\boldsymbol{\theta}/t$).

$$\begin{aligned}
\boldsymbol{\theta} = \boldsymbol{\theta} - \eta\boldsymbol{M}_T\boldsymbol{\mu}_T &= \boldsymbol{\theta} - \eta\left[\boldsymbol{M}_{T-1} - \frac{\boldsymbol{M}_{T-1}\mathbf{q}_{T-1}\mathbf{q}_{T-1}^\top\boldsymbol{M}_{T-1}}{1 + \mathbf{q}_{T-1}^\top\mathbf{M}_{T-1}\mathbf{q}_{T-1}}\right](\boldsymbol{\mu}_{T-1} + \boldsymbol{\xi}_{T-1}) \\
&= \boldsymbol{\theta} - \eta\boldsymbol{M}_{T-1}\boldsymbol{\mu}_{T-1} - \eta\boldsymbol{M}_{T-1}\boldsymbol{\xi}_{T-1} + \frac{\boldsymbol{M}_{T-1}\mathbf{q}_{T-1}\mathbf{q}_{T-1}^\top\boldsymbol{M}_{T-1}}{1 + \mathbf{q}_{T-1}^\top\mathbf{M}_{T-1}\mathbf{q}_{T-1}}\boldsymbol{\mu}_{T-1} \\
&\quad + \frac{\boldsymbol{M}_{T-1}\mathbf{q}_{T-1}\mathbf{q}_{T-1}^\top\boldsymbol{M}_{T-1}}{1 + \mathbf{q}_{T-1}^\top\mathbf{M}_{T-1}\mathbf{q}_{T-1}}\boldsymbol{\xi}_{T-1}.
\end{aligned}$$

We can further expand $\boldsymbol{M}_{T-1}\boldsymbol{\mu}_{T-1}$ as:

$$\boldsymbol{M}_{T-1}\boldsymbol{\mu}_{T-1} = \left[\boldsymbol{M}_{T-2} - \frac{\boldsymbol{M}_{T-2}\mathbf{q}_{T-2}\mathbf{q}_{T-2}^\top\boldsymbol{M}_{T-2}}{1 + \mathbf{q}_{T-2}^\top\mathbf{M}_{T-2}\mathbf{q}_{T-2}}\right](\boldsymbol{\mu}_{T-2} + \boldsymbol{\xi}_{T-2})$$

$$= \boldsymbol{M}_{T-2}\boldsymbol{\mu}_{T-2} + \boldsymbol{M}_{T-2}\boldsymbol{\xi}_{T-2} - \frac{\boldsymbol{M}_{T-2}\mathbf{q}_{T-2}\mathbf{q}_{T-2}^\top\boldsymbol{M}_{T-2}}{1 + \mathbf{q}_{T-2}^\top\mathbf{M}_{T-2}\mathbf{q}_{T-2}}\boldsymbol{\mu}_{T-2} - \frac{\boldsymbol{M}_{T-2}\mathbf{q}_{T-2}\mathbf{q}_{T-2}^\top\boldsymbol{M}_{T-2}}{1 + \mathbf{q}_{T-2}^\top\mathbf{M}_{T-2}\mathbf{q}_{T-2}}\boldsymbol{\xi}_{T-2}$$

and again we can further expand $\boldsymbol{M}_{T-2}\boldsymbol{\mu}_{T-2}$ as:

$$\boldsymbol{M}_{T-2}\boldsymbol{\mu}_{T-2} = \left[\boldsymbol{M}_{T-3} - \frac{\boldsymbol{M}_{T-3}\mathbf{q}_{T-3}\mathbf{q}_{T-3}^\top\boldsymbol{M}_{T-3}}{1 + \mathbf{q}_{T-3}^\top\mathbf{M}_{T-3}\mathbf{q}_{T-3}}\right](\boldsymbol{\mu}_{T-3} + \boldsymbol{\xi}_{T-3})$$

$$= \boldsymbol{M}_{T-3}\boldsymbol{\mu}_{T-3} + \boldsymbol{M}_{T-3}\boldsymbol{\xi}_{T-3} - \frac{\boldsymbol{M}_{T-3}\mathbf{q}_{T-3}\mathbf{q}_{T-3}^\top \boldsymbol{M}_{T-3}}{1+\mathbf{q}_{T-3}^\top \mathbf{M}_{T-3}\mathbf{q}_{T-3}}\boldsymbol{\mu}_{T-3} - \frac{\boldsymbol{M}_{T-3}\mathbf{q}_{T-3}\mathbf{q}_{T-3}^\top \boldsymbol{M}_{T-3}}{1+\mathbf{q}_{T-3}^\top \mathbf{M}_{T-3}\mathbf{q}_{T-3}}\boldsymbol{\xi}_{T-3}.$$

We repeat these steps. In the last step we expand $\boldsymbol{M}_1\boldsymbol{\mu}_1$. We can combine these results and write the following update rule:

$$\boldsymbol{\theta} = \boldsymbol{\theta} - \eta \boldsymbol{M}_T\boldsymbol{\mu}_T = \boldsymbol{\theta} - \eta \boldsymbol{M}_0\boldsymbol{\mu}_0 - \eta \sum_{c=0}^{T-1}\left[\boldsymbol{M}_c\boldsymbol{\xi}_c - \frac{\boldsymbol{M}_c\mathbf{q}_c\mathbf{q}_c^\top \boldsymbol{M}_c}{1+\mathbf{q}_c^\top \mathbf{M}_c\mathbf{q}_c}\boldsymbol{\mu}_c - \frac{\boldsymbol{M}_c\mathbf{q}_c\mathbf{q}_c^\top \boldsymbol{M}_c}{1+\mathbf{q}_c^\top \mathbf{M}_c\mathbf{q}_c}\boldsymbol{\xi}_c\right]$$

$$= \boldsymbol{\theta} - \eta \sum_{c=0}^{T-1}\left[\left(\boldsymbol{M}_c - \frac{\boldsymbol{M}_c\mathbf{q}_c\mathbf{q}_c^\top \boldsymbol{M}_c}{1+\mathbf{q}_c^\top \mathbf{M}_c\mathbf{q}_c}\right)\boldsymbol{\xi}_c - \frac{\boldsymbol{M}_c\mathbf{q}_c\mathbf{q}_c^\top \boldsymbol{M}_c}{1+\mathbf{q}_c^\top \mathbf{M}_c\mathbf{q}_c}\boldsymbol{\mu}_c\right]$$

The last inequality comes from the fact that $\boldsymbol{\mu}_0$ is initialized as $\boldsymbol{\mu}_0 = \mathbf{0}$. We have thus rewritten the batch majorization update rule for the parameters as an iterative summation over the data. Analogous to the conversion of a batch gradient descent algorithm into SGD, the most natural way to convert the batch bound majorization algorithm to its stochastic version is to interleave updates of the parameter $\boldsymbol{\theta}$ rather than waiting for the full summation to terminate (after an epoch) before allowing $\boldsymbol{\theta}$ to update. This permits us to now write a fully stochastic update rule on the parameter vector $\boldsymbol{\theta}$ (in original notation):

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \eta_i \left[\left(\boldsymbol{M}_j(\boldsymbol{\theta}_i) - \frac{\boldsymbol{M}_j(\boldsymbol{\theta}_i)\mathbf{q}_j(\boldsymbol{\theta}_i)\mathbf{q}_j(\boldsymbol{\theta}_i)^\top \boldsymbol{M}_j(\boldsymbol{\theta}_i)}{1+\mathbf{q}_j(\boldsymbol{\theta}_i)^\top \mathbf{M}_j(\boldsymbol{\theta}_i)\mathbf{q}_j(\boldsymbol{\theta}_i)}\right)\boldsymbol{\xi}_j(\boldsymbol{\theta}_i) \right.$$
$$\left. - \frac{\boldsymbol{M}_j(\boldsymbol{\theta}_i)\mathbf{q}_j(\boldsymbol{\theta}_i)\mathbf{q}_j(\boldsymbol{\theta}_i)^\top \boldsymbol{M}_j(\boldsymbol{\theta}_i)}{1+\mathbf{q}_j(\boldsymbol{\theta}_i)^\top \mathbf{M}_j(\boldsymbol{\theta}_i)\mathbf{q}_j(\boldsymbol{\theta}_i)}\boldsymbol{\mu}_j(\boldsymbol{\theta}_i)\right]$$

Algorithm 2 Stochastic Bound

```
Input: $\lambda \in \mathbb{R}^+$, $\eta$
Initialize: $\boldsymbol{\theta} \in \mathbb{R}^d$, $\boldsymbol{\phi} = zeros(d)$, $\boldsymbol{M} = \frac{1}{\lambda}\boldsymbol{I}$, $\boldsymbol{\mu} = \mathbf{0}$
while not converged {
    select data point $j$
    $z \to 0^+$;  $\boldsymbol{g} = \mathbf{0}$
    For each $y \in \Omega$  {
        $\alpha = h_j(y)\exp(\boldsymbol{\theta}^\top \boldsymbol{f}_j(y))$;  $\boldsymbol{l} = \boldsymbol{f}_j(y) - \boldsymbol{g}$;  $\beta = \frac{\tanh(\frac{1}{2}\log(\alpha/z))}{2\log(\alpha/z)}$;  $\kappa = \frac{\alpha}{z+\alpha}$
        $\boldsymbol{\xi} = \kappa\boldsymbol{l} - \boldsymbol{f}_j + \lambda\boldsymbol{\theta}/t$
        $\boldsymbol{N} \;\; = \frac{\boldsymbol{M}\beta\boldsymbol{l}\boldsymbol{l}^\top\boldsymbol{M}}{1+\beta\boldsymbol{l}^\top\boldsymbol{M}\boldsymbol{l}}$
        $\boldsymbol{M} -= \boldsymbol{N}$
        $\boldsymbol{\phi} \;\; += \boldsymbol{M}\boldsymbol{\xi} - \boldsymbol{N}\boldsymbol{\mu}$
        $\boldsymbol{g} \;\; += \kappa\boldsymbol{l}$
        $\boldsymbol{\mu} \;\; += \boldsymbol{\xi}$
        $z \;\; += \alpha$  }
    $\boldsymbol{\theta} \;\; -= \eta\boldsymbol{\phi}$  }
```

The stochastic bound majorization algorithm also readily admits mini-batches. Algorithm 2 captures the full-rank version of the proposed algorithm (we always use constant step size $\eta = \frac{1}{t}$).

We have also investigated many other potential variants of Algorithm 2 including heuristics borrowed from other stochastic algorithms in the literature [12]. Some heuristics involved using memory to store previous values of updates, gradients and second order matrix information. Remarkably, all such heuristics and modifications slowed down the convergence of Algorithm 2.

On caveat remains. The computational complexity of the proposed stochastic bound majorization method is $\mathcal{O}(nd^2)$ per iteration which is less appealing than the $\mathcal{O}(nd)$ complexity of SGD. This shortcoming is resolved in the next subsection.

3.2 Low-rank version

We next provide a low-rank version of Algorithm 2 to maintain a $\mathcal{O}(nd)$ run-time per stochastic update. Consider the update on $\boldsymbol{\phi}$ inside the loop over y in Algorithm 2. This update can be rewritten as

$$\boldsymbol{\phi} += \boldsymbol{M}\boldsymbol{\xi} - \boldsymbol{N}\boldsymbol{\mu} = \boldsymbol{M}\boldsymbol{\xi} - (\boldsymbol{M}_{old} - \boldsymbol{M})\boldsymbol{\mu} = (\boldsymbol{\Sigma})^{-1}\boldsymbol{\xi} - ((\boldsymbol{\Sigma}_{old})^{-1} - (\boldsymbol{\Sigma})^{-1})\boldsymbol{\mu},$$

where $\mathbf{M}_{old}$ and $\mathbf{\Sigma}_{old}$ are the matrices that, after being updated, become $\mathbf{M}$ and $\mathbf{\Sigma}$ respectively: $\mathbf{M} = \mathbf{M}_{old} - \mathbf{N}_{old}$, $\mathbf{M}_{old} = \mathbf{\Sigma}_{old}^{-1}$ and $\mathbf{\Sigma} = \mathbf{\Sigma}_{old} + \mathbf{q}_{old}\mathbf{q}_{old}^{\top} = \mathbf{\Sigma}_{old} + \beta_{old}\mathbf{l}_{old}\mathbf{l}_{old}^{\top}$ (rank 1 update). We can store the matrix $\mathbf{\Sigma}$ using a low-rank representation $\mathbf{V}^{\top}\mathbf{S}\mathbf{V} + \mathbf{D}$, where k is a rank ($k << d$), $\mathbf{V} \in \mathbb{R}^{k \times d}$ is orthonormal, $\mathbf{S} \in \mathbb{R}^{k \times k}$ is positive semi-definite and $\mathbf{D} \in \mathbb{R}^{d \times d}$ is non-negative diagonal. We can directly update $\mathbf{V}$, $\mathbf{S}$ and $\mathbf{D}$ online rather than incrementing matrix $\mathbf{\Sigma}$ by a rank 1 update. In the case of batch bound majorization method this still guarantees an overall upper bound [27]. We directly apply this technique as well in our stochastic setting. Due to space constraints we will not present this technique (we refer the reader to [27]). Given a low-rank version of the matrix, we use the Woodbury formula to invert it in each iteration: $\mathbf{\Sigma}^{-1} = \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{V}^{\top}(\mathbf{S}^{-1} + \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^{\top})^{-1}\mathbf{V}\mathbf{D}^{-1}$. That leads to a low-rank version of Algorithm 2, which requires only $\mathcal{O}(knd)$ work per iteration which is pseudo-linear in dimension if k is assumed to be a logarithmic or constant function of d.

4 A motivating example

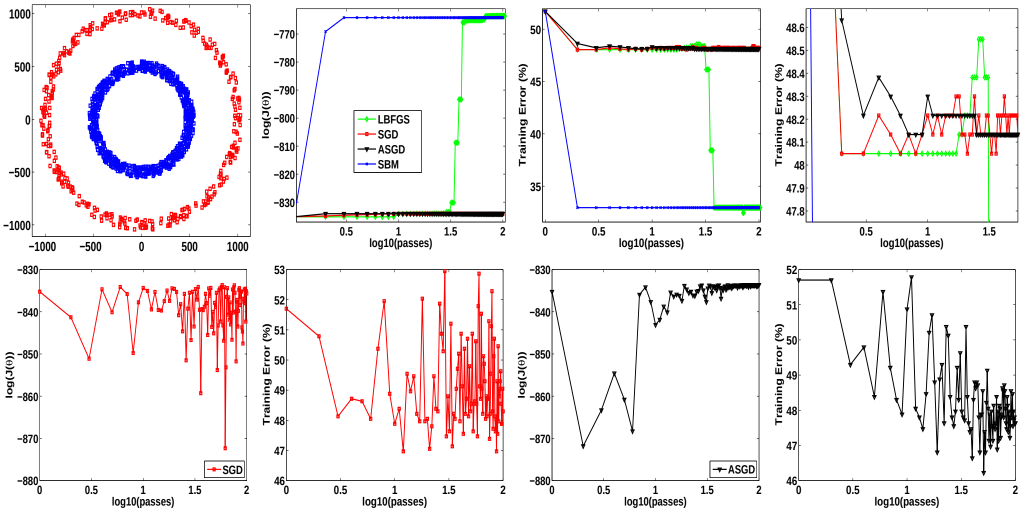


Figure 1: A comparison of LBFGS, SGD, ASGD and SBM for l_2-regularized logistic regression. **From left to right, first row:** the data-set, training log-likelihood and error (original and zoomed) vs. passes through the data for LBFGS, SGD ($\eta_0 = 10^{-8}$, $m = 1$), ASGD ($\eta_0 = 10^{-5}$, $\tau = 1$, $m = 1$) and SBM ($\eta_0 = \frac{1}{t}$, $m = 1$), **second row**: SGD training log-likelihood and error ($\eta_0 = 10^{-7}$) and ASGD training log-likelihood and error ($\eta_0 = 10^{-4}$, $\tau = 1$) vs. passes through the data. $\lambda = 10^1$.

Consider Figure 1 which is an example of a binary classification problem which exposes some of difficulties with SGD. Intuitively, in this example, the gradients in the SGD update rule will point in almost random directions which could lead to very slow progress. Four training algorithms will be compared: SGD, ASGD, stochastic bound majorization algorithm (SBM) and LBFGS. We have tried several parameter settings for SGD and ASGD. The range of tested step size η_0 was as broad as $[10^{-12}, 1]$. The SGD method however exhibits high instability until the step size is reduced to an unreasonably small value such as 10^{-8} (for comparison we also show SGD performance for $\eta_0 = 10^{-7}$). The reason for is that this is a highly symmetric and non-linearly separable data-set. Therefore, the information captured in the gradients is extremely noisy which causes SGD to oscillate and fail to converge in practice. For ASGD we obtained the best and stable result for $\eta_0 = 10^{-5}$ (for comparison we also show ASGD performance for $\eta_0 = 10^{-4}$) and $\tau = 1$ (larger values of τ weaken performance). For both methods we tested the mini-batch size m from 1 (a single data point) to 10 and noticed no meaningful difference. Clearly both SGD and ASGD are stuck in solutions that are only slightly better than random guessing. Meanwhile SBM (with $m = 1$ and a constant step size $\eta_0 = \frac{1}{t}$) finds the same solution as a batch LBFGS method. It does so with a single pass through the data and simultaneously outperforming the competitor methods. We emphasize that all methods used as comparators to our algorithm were well-tuned, i.e. the initial gain η_0 is set as high as possible for each method while maintaining stability.

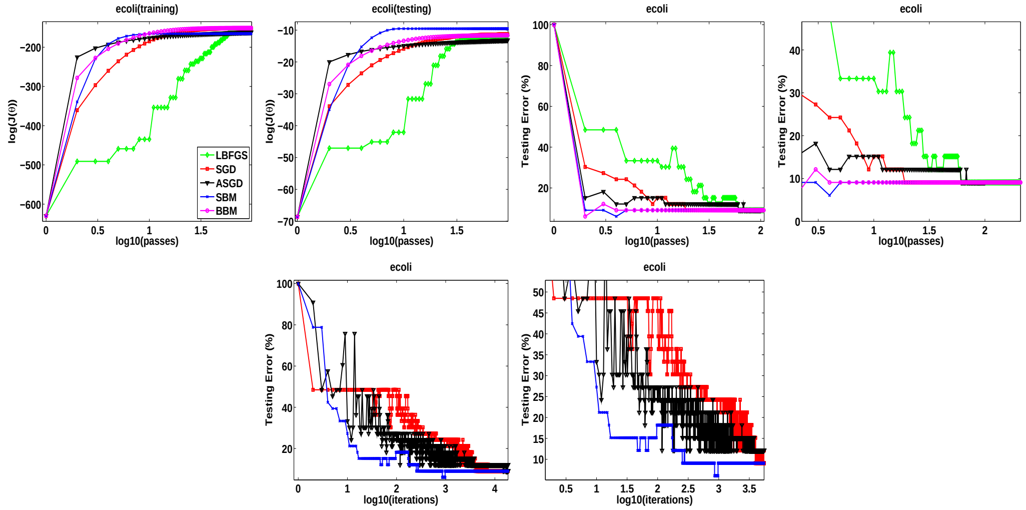


Figure 2: A comparison of LBFGS, SGD, ASGD, SBM and BBM for l_2-regularized logistic regression. **From left to right, first row:** training and testing log-likelihood and testing error (original and zoomed) vs. passes through the data for LBFGS, SGD ($\eta_0 = 10^{-2}$, $m = 10$), ASGD ($\eta_0 = 10^{-1}$, $\tau = 500$, $m = 10$), SBM ($\eta_0 = \frac{1}{t}$, $m = 1$) and BBM on the ecoli data-set, **second row**: testing error (original and zoomed) vs. iterations for LBFGS, SGD and ASGD on the same experiment.

Next, we focus on the ecoli UCI data-set (http://archive.ics.uci.edu/ml/), a simple small-scale classification problem. We compare LBFGS, batch bound majorization method (BBM), SGD, ASGD and SBM. Results are summarized in Figure 2. BBM, which was already shown to outperform leading batch methods [27], performs comparably to ASGD and SGD. The only method that beats BBM is SBM. In the first row of Figure 2, we plot the objective with respect to the number of passes through the data. However, looking more closely at the objective with respect to each iteration (where a single iteration corresponds to a single update of the parameter vector), we note an interesting property of SBM: it clearly remains monotonic in its convergence despite its stochastic nature. This is in contrast to SGD and ASGD which (as expected) fluctuate much more noisily.

Note that, in all experiments in this section and the next, 90% of the data is used for training and the rest for testing, the results are averaged over 10 random initializations close to the origin and the regularization value λ is chosen through crossvalidation. All methods were implemented in C++ using the mex environment under Matlab.

5 Experiments

We next evaluate the performance of the new algorithm empirically. We compare SGD, ASGD and SBM for l_2-regularized logistic regression on the Mnist*, gisette*, SecStr†, digitl† and Text† data-sets[1]. We show two variants of the SBM algorithm: full-rank (on SecSTR and Mnist) and low-rank (on the remaining data-sets). For the experiments with full-rank SBM, we plot the testing log-likelihood and error versus passes through the data (epoch iterations). For the experiments with low-rank SBM, we plot the likelihood versus cpu time. For SGD and ASGD we tested mini-batches of size from $m = 1$ to $m = 10$ and chose the best setting. For the Mnist data-set we explored mini-batches of up to $m = 100$ to achieve optimal SGD behavior. For SBM we always simply used $m = 1$. For each data-set, Figure 3 reports the optimal step size η_0 for SGD and ASGD and the optimal parameter τ for ASGD (if it was necessary). For the full-rank version of SBM we always use $\eta_0 = \frac{1}{t}$, however for its low-rank version (where we simply assumed $k = 1$) we

[1] Downloaded from *http://yann.lecun.com/exdb/mnist/, *http://archive.ics.uci.edu/ml/ and †http://olivier.chapelle.cc/ssl-book/benchmarks.html

tuned η_0. The chosen value of η_0 is also reported in Figure 3. Clearly, SBM is less prone to overfitting and achieves higher testing likelihood than SGD and ASGD as well as lower testing error. Simultaneously, SBM exhibits the fastest convergence in terms of the number of passes through the data till convergence. Furthermore, low-rank SBM had the fastest convergence in terms of cpu time, outperforming the leading stochastic first-order methods (SGD and ASGD) as shown in the plots of likelihood over time.

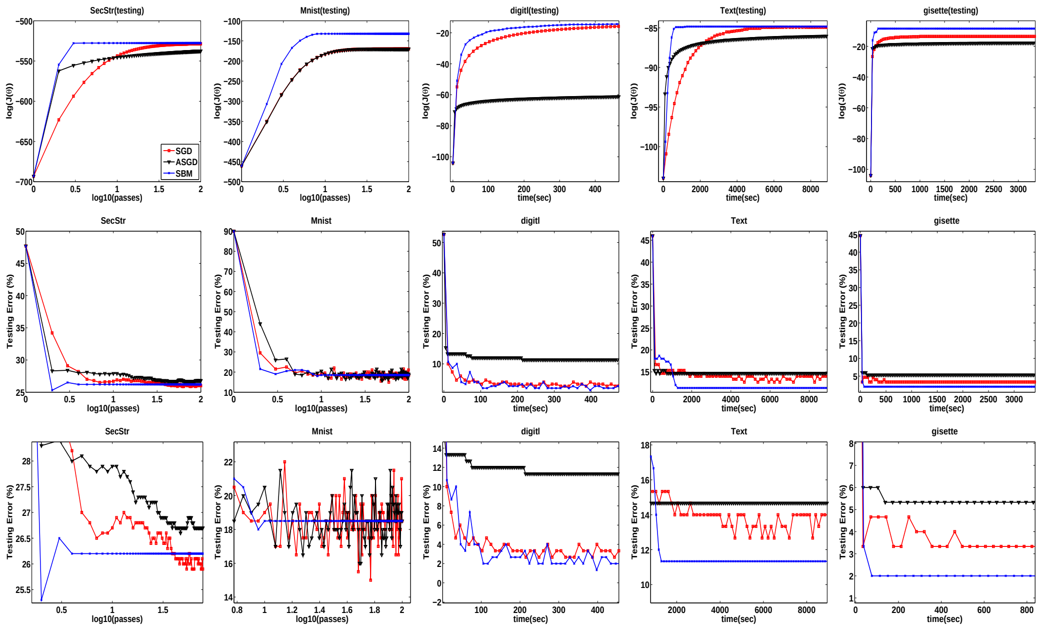


Figure 3: A comparison of SGD, ASGD and SBM for l_2-regularized logistic regression. **From top to bottom:** testing log-likelihood and testing error (original and zoomed) vs. passes through the data for data-sets: **SecStr** ($t = 83679$, $n = 2$, $d = 632$, $\lambda = 10^1$, SGD ($\eta_0 = 10^{-4}$, $m = 10$), ASGD ($\eta_0 = 10^{-1}$, $m = 10$) SBM ($\eta_0 = \frac{1}{t}$, $m = 1$, full-rank)), **Mnist** ($t = 10000$, $n = 10$, $d = 510$, $\lambda = 10^{-2}$, SGD ($\eta_0 = 10^{-1}$, $m = 100$), ASGD ($\eta_0 = 10^{-1}$, $\tau = 10^5$, $m = 100$) SBM ($\eta_0 = \frac{1}{t}$, $m = 1$, full-rank)), **digitl** ($t = 1500$, $n = 2$, $d = 1448$, $\lambda = 10^1$, SGD ($\eta_0 = 10^{-4}$, $m = 10$), ASGD ($\eta_0 = 10^{-3}$, $\tau = 10^2$, $m = 10$) SBM ($\eta_0 = 50 \cdot \frac{1}{t}$, $m = 1$, $k = 1$)), **Text** ($t = 1500$, $n = 2$, $d = 23922$, $\lambda = 10^1$, SGD ($\eta_0 = 10^{-4}$, $m = 10$), ASGD ($\eta_0 = 10^{-2}$, $\tau = 10^3$, $m = 10$) SBM ($\eta_0 = \frac{1}{t}$, $m = 1$, $k = 1$)) and **gisette** ($t = 1500$, $n = 2$, $d = 10002$, $\lambda = 10^0$, SGD ($\eta_0 = 10^{-3}$, $m = 10$), ASGD ($\eta_0 = 10^{-1}$, $\tau = 10^1$, $m = 10$) SBM ($\eta_0 = 100 \cdot \frac{1}{t}$, $m = 1$, $k = 1$)).

6 Conclusion

We have proposed a new stochastic bound majorization method for optimizing the partition function of log-linear models that uses second-order curvature through a global bound (rather than a local Hessian). The method is obtained by applying Sherman-Morrison to the batch update rule to convert it into an iterative summation over the data which can easily be made stochastic by interleaving parameter updates. This (full-rank) stochastic method requires no parameter tuning. A low-rank version of this stochastic update rule makes this effectively second-order method remain linear in the dimensionality of the data. We showed experimentally that the method has significant advantage over the state-of-the-art first-order stochastic methods like SGD and ASGD making majorization competitive in both stochastic and batch settings [27]. Stochastic bound majorization achieves convergence in fewer iterations, in less computation time (when using the low-rank version), and with better final solutions. Future work will involve providing theoretical guarantees for the method as well as application to deep architectures with cascaded linear combinations of soft-max functions.

References

[1] L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[2] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, April 1988.

[3] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[4] L. Bottou. Stochastic gradient learning in neural networks. In *Proceedings of Neuro-Nmes*. 1991.

[5] Y. Le Cun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *NIPS*, 2012.

[7] B. Kingsbury, T. N. Sainath, and H. Soltau. Scalable minimum Bayes risk training of deep neural network acoustic models using distributed Hessian-free optimization. In *INTERSPEECH*, 2012.

[8] S. V. N. Vishwanathan, N. N. Schraudolph, M. W. Schmidt, and K. P. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML*, 2006.

[9] N. Le Roux and A. W. Fitzgibbon. A fast natural Newton method. In *ICML*, 2010.

[10] C. Wang and D. M. Blei. Truncation-free online variational inference for Bayesian nonparametric models. In *NIPS*, 2012.

[11] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[12] N. Le Roux, M. W. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*, 2012.

[13] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855, July 1992.

[14] P. Tseng. An incremental gradient(-projection) method with momentum term and adaptive stepsize rule. *SIAM J. on Optimization*, 8(2):506–531, February 1998.

[15] Y. Nesterov. Primal-dual subgradient methods for convex problems. *Math. Program.*, 120(1):221–259, 2009.

[16] H. Kesten. Accelerated stochastic approximation. *Annals of Mathematical Statistics*, 29(1):41–59, 1958.

[17] D. Blatt, A. O. Hero, and H. Gauchman. A convergent incremental gradient method with a constant step size. *SIAM Journal on Optimization*, 18(1):29–51, 2007.

[18] L. Xiao. Dual averaging methods for regularized stochastic learning and online optimization. In *NIPS*, 2009.

[19] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. In *ICML*, 2011.

[20] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *NIPS*, 2007.

[21] N. N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *ICANN*, 1999.

[22] N. N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-Newton method for online convex optimization. In *AISTATS*, 2007.

[23] S.-I. Amari, H. Park, and K. Fukumizu. Adaptive method of realizing natural gradient learning for multilayer perceptrons. *Neural Comput.*, 12(6):1399–1409, June 2000.

[24] A. Bordes, L. Bottou, and P. Gallinari. Sgd-qn: Careful quasi-Newton stochastic gradient descent. *J. Mach. Learn. Res.*, 10:1737–1754, December 2009.

[25] J. Martens. Deep learning via Hessian-free optimization. In *ICML*, 2010.

[26] M. P. Friedlander and M. W. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM J. Scientific Computing*, 34(3), 2012.

[27] T. Jebara and A. Choromanska. Majorization for CRFs and latent likelihoods. In *NIPS*, 2012.

[28] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[29] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7):1527–1554, 2006.

[30] L.R. Bahl, M. Padmanabhan, D. Nahamoo, and P. S. Gopalakrishnan. Discriminative training of Gaussian mixture models for large vocabulary speech recognition systems. In *ICASSP*, 1996.

[31] A. Berger. The improved iterative scaling algorithm: A gentle introduction. *Technical report, Carnegie Mellon University*, 1997.

[32] J. De Leeuw and W. J. Heiser. Convergence of correction matrix algorithms for multidimensional scaling. *chapter Geometric representations of relational data, Mathesis Press*, pages 735–752, 1977.

[33] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[34] R. Salakhutdinov and S. T. Roweis. Adaptive overrelaxed bound optimization methods. In *ICML*, 2003.